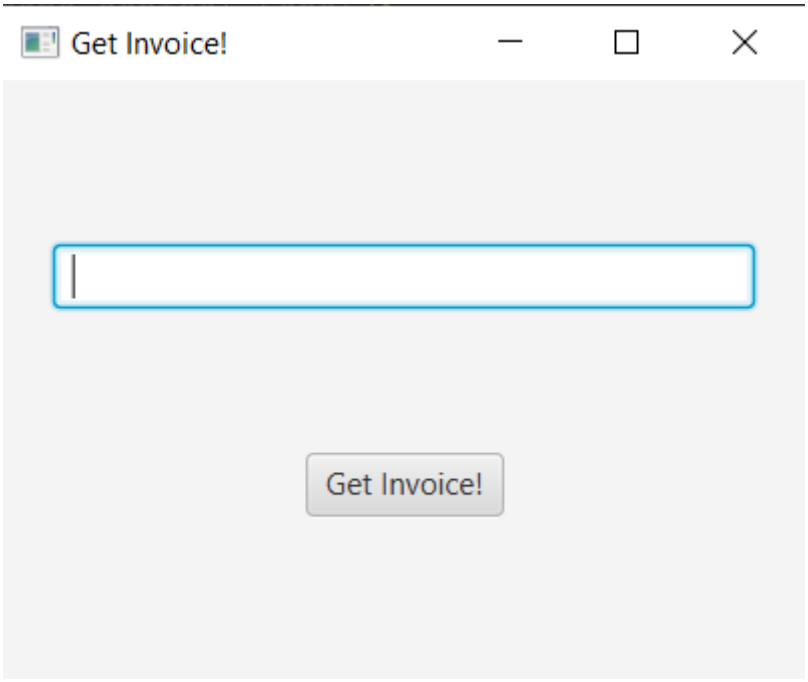
Fuel Station

Contents

Applications

Javafx



When the user starts the Java FX application, he can input a customer ID. Then the button “Get Invoice!” has to be clicked, so a request get sent to the Spring Boot application. The user gets feedback if the file got created or not.

Spring Boot

Handles requests of Java FX application. Sends message to “dispatcherQueue” with the customer ID.

Collection Dispatcher

Consumes messages from “dispatcherQueue”. Checks in the Stations DB which stations are available and takes this information (adds the customer ID) and sends a message into the “collectorQueue”.

Data Collector

Consumes messages from “collectorQueue”. Checks against every DB of each Station if there are any charges for the customer ID. Collects necessary data and creates a message. The message is then sent into the “recieverQueue”.

Collection Receiver

Consumes messages from “recieverQueue”. Calculates sum of sent data. Sends message to the “generateQueue”.

PDF Generator

Consumes messages from the “generateQueue”. Checks consumer ID against Consumer DB to get full name of customer. Generates Invoice PDF (FuelStation/Worker/PDFGenerator).

Technical workflow + assumptions

The java FX application sends Post Request http://localhost:8080/invoices/{ID}

Spring Boot applications handels Post Request in Method “getID”. Takes ID and sends it in “dispatcherQueue”.

Collections Dispatcher takes message from “dispatcherQueue”. In method “executeInternal” of class DispatcherService it connects to StationDB (localhost:30002). Then it executes a sql Statement “select * from stations”. Then it creates a message with following pattern.
The message has two main parts which a separated with a “#”.
The first part is just the customer ID. The second part are Station ID and the db_url, which are separated with an “@”. The second part gets repeated for each available Station. The stations are separated by a “|”.

Example:
2#1@localhost:30011|2@localhost:30012|3@localhost:30013|
Two main parts separated with “#”.
Customer ID → 2
Information regarding each station →1@localhost:30011|2@localhost:30012|3@localhost:30013|
The Stations are separated by “|” (so we have three stations).

Message gets send to “collectorQueue”.

Data Collector consumes message from “collectorQueue”. In method “executeInternal” of class CollectorService it splits up the received message to get the needed information. It connects to each DB of each Stations and executes a sql statement “SELECT * FROM charge where customer_id ={customerID}”. A message get created, starting with the customer_id and then with all the values of all the databases of column “kwh” from the executed sql statement.
Example:
1#0|10.8|10.6|49.7|32.9|41.3|43.5|31.7|13.8|19.1|25.9|37.4|10.7|27.2|20.5|45.8|
customer_id → 1
kwh values (separated by “|”) →
0|10.8|10.6|49.7|32.9|41.3|43.5|31.7|13.8|19.1|25.9|37.4|10.7|27.2|20.5|45.8|

The 0 at the beginning is for error handling and making sure nothing crashes. Details follow.
Message gets send to “receiverQueue”.

Collection Reciever consumes message from “recieverQueue”. In method “executeInternal” of class RecieverService it splits up the received message to get the needed information. It sums up all the “kwh” values and creates a message.
Example:
1#420.9
consumer id → 1
sum of values → 420.9

Message gets send to “generateQueue”.

PDF Generator consumes message from “generateQueue”. In method “executeInternal” of class PDFGenerator, it splits up the received message to get the needed information. It connects to the customer Database an executes the following select statement “SELECT * FROM customer where id ={customerID}”. Doe to that, first and second name of customer is available. The invoice gets

generated as an PDF. First Page contains a greeting. Second page the invoice information → sum of kwh of the customer, cost per kwh (assumption) and cost (kwh of customer * cost per kwh).

Lastly, the java fx application runs in a loop a get Request which again is handled by the Spring Boot application. The Spring boot applications checkes if the pdf Invoice for the customer ID already got created. “IS HERE!!!!” is send to the java fx application when the pdf exists, “Not coming..Sorry” if it is not there (yet). Java fx is printing the message in the console. The Get Request is executed 7 time, with a delay of 2 seconds.

Error Handling

Although the application is quite simple, there can occur problems, like user enters a customer id which does not exist. This is handled by the application. Especially in the Data Collector, because then the sql statement can retrieve no values or will fail, because there are letters in the customer id. This is handled. It just send a standard message further looking like this:
{customer_id}#0

The Collection Reciever behaves similar.
The customer id problem would also occur when the PDF generator checks the customer ID against the Customer DB. This is also handled. A PDF will be created with the text: “"KEINE DATEN GEFUNDEN ZU CUSTOMER_ID: {customerID}”

Testing

The Spring Boot application also has an implemented Unit-Test. The method “hereOrNot” checks if the defined file exists or not. So, it can be checked, if the correct feedback is given to the java FX application.